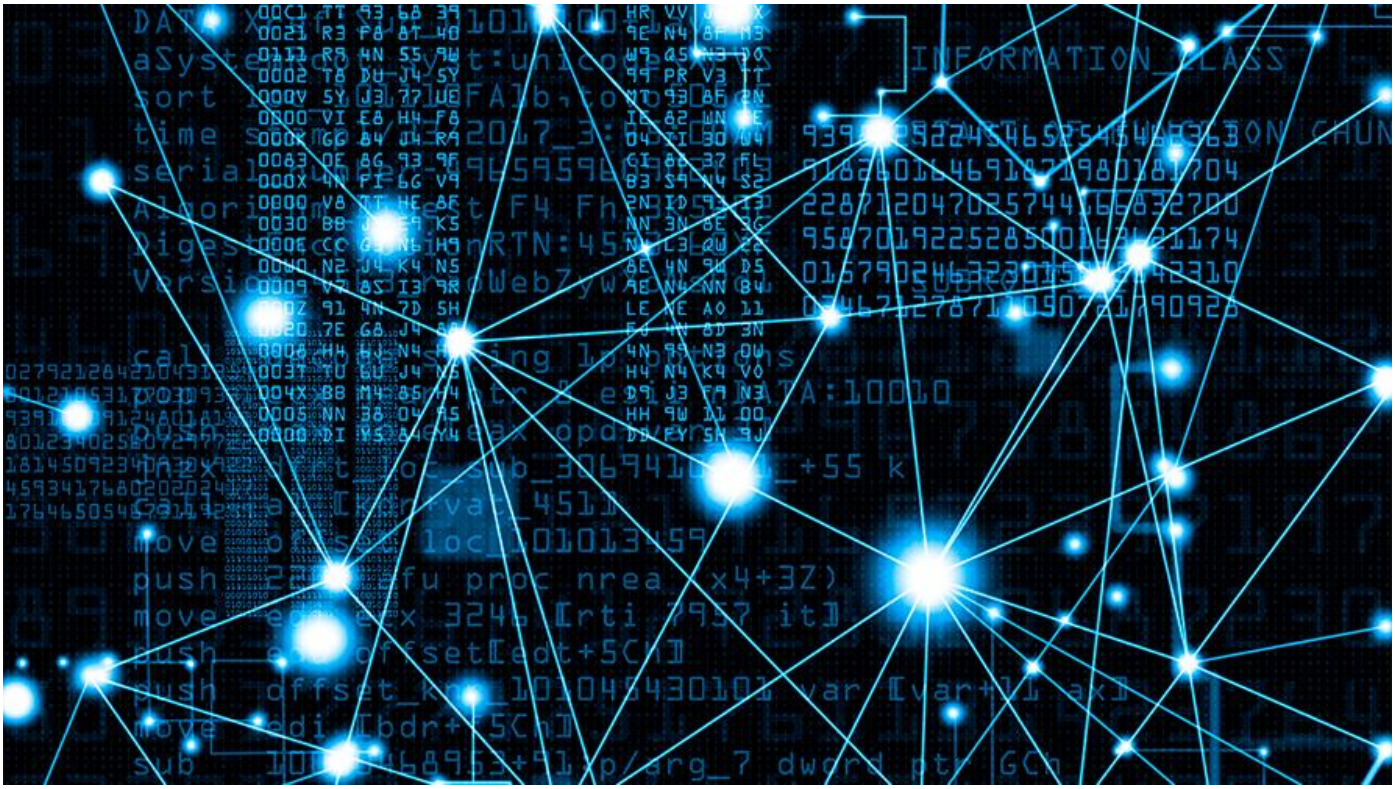
ΑΛΓΟΡΙΘΜΟΙ ΚΑΙ ΠΟΛΥΠΛΟΚΟΤΗΤΑ

2Η ΣΕΙΡΑ ΓΡΑΠΤΩΝ ΑΣΚΗΣΕΩΝ



DECEMBER 20, 2021

ΘΟΔΩΡΗΣ ΑΡΑΠΗΣ – EL18028

Άσκηση 1

Έχουμε:

$p[n]$ πίνακα με τις συντεταγμένες κάθε στοιχείου p:

$p[k] = (x_k, y_k),\ k \in [1, n]$ ((x_k, y_k) συντεταγμένες στο επίπεδο)

δίσκοι ακτίνας r,

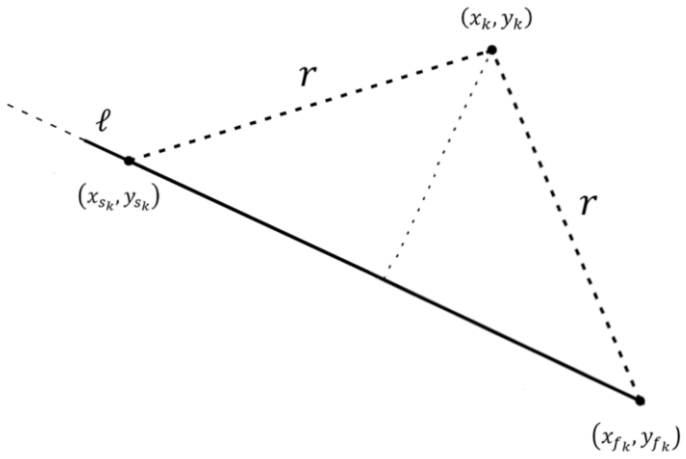
ευθεία μήκους ℓ

Ο αλγόριθμος θα είναι ο εξής:

Σε $O(n)$ δημιουργώ έναν πίνακα διαστημάτων $d[n]$ της μορφής:

$$d[k] = (s_k, f_k), \qquad x_{s_k} \leq x_{f_k}$$

όπου $s_k = (x_{s_k}, y_{s_k}), f_k = (x_{f_k}, y_{f_k})$ τα δύο σημεία της ευθείας ℓ που απέχουν απόσταση r από το σημείο $p[k]$.



Ταξινομούμε σε $O(n \log n)$ τον πίνακα $d[n]$ ως προς τα σημεία f_k, σε αύξουσα σειρά.

Οι δίσκοι θα τοποθετούνται με κέντρο τα σημεία f_k. Με αυτόν τον τρόπο το σημείο $p[k]$, καθώς και όλα τα σημεία $p[i],\ i > k$ για τα οποία ισχύει $s_i \leq f_k$ θα καλύπτονται από τον δίσκο στο f_k. Άρα αρκεί να ξεκινήσουμε από το $d[0]$ και να διατρέξουμε όλον πίνακα σε $O(n)$, εκτελώντας τις εξής ενέργειες:

- Αρχικές τιμές $disks = [f_1]$ $(\lambda ίστα)$,
- $Για\ i\ από\ 2\ εώς\ n\ επανέλαβε$:

$$Αν\ s_i > disks[last]\ (τελευταίο\ στοιχείο\ της\ λίστας)\ τότε:$$

- $βάλε\ στο\ τέλος\ της\ λίστας\ το\ f_i$

$$Αλλιώς\ συνέχισε\ την\ επανάληψη$$

Στο τέλος θα έχουμε μία λίστα $disks$ με τα σημεία στα οποία πρέπει να τοποθετηθούν οι δίσκοι με βέλτιστο τρόπο.

Ορθότητα:

Έστω ότι ο παραπάνω άπληστος αλγόριθμος επιστρέφει μία λίστα $disks_{greedy} = [g_1, g_2, \dots, g_d]$ με τα σημεία των δίσκων. Έστω ακόμη ότι η βέλτιστη λύση είναι μία λίστα $disks_{optimal} = [o_1, o_2, \dots, o_p]$. Για να καλύπτεται το $p[1]$ θα πρέπει ο πρώτος δίσκος να τοποθετηθεί οπωσδήποτε στο διάστημα $[s_1, f_1]$. Αφού $g_1 = f_1$ τότε θα ισχύει $o_1 \leq g_1$. Συνεπώς ο δίσκος του άπληστου αλγόριθμου θα καλύπτει εκτενέστερη (ή ίση) περιοχή προς τα δεξιά από ότι ο δίσκος της βέλτιστης λύσης. Επομένως, όταν κάποιο σημείο $p[i]$ δεν καλύπτεται από τον δίσκο στο g_1 $(s_i > g_1)$, τότε ο επόμενος δίσκος θα πρέπει να τοποθετηθεί οπωσδήποτε στο διάστημα $[s_i, f_i]$. Αφού $g_2 = f_i$, όμοια με πριν, θα έχουμε $o_2 \leq g_2$. Οπότε επαγωγικά ισχύει ότι $o_j \leq g_j$.

Τέλος, είναι προφανές πως $d = p$, μιας και αν $d > p$ και $o_p \leq g_p$ τότε θα υπήρχε κάποιο σημείο με $p + 1 \leq d$: $s_{p+1} > g_p \geq o_p$. Άρα $o_p < s_{p+1}$ που σημαίνει πως υπάρχει κάποιο ακάλυπτο σημείο. Άτοπο.

Επομένως η άπληστη λύση είναι βέλτιστη.

Χρονική πολυπλοκότητα: $O(n \log n)$

Άσκηση 2

1.

Αρχικά ορίζουμε μία οποιαδήποτε σειρά δρομολόγησης ως q.

Οι σχέσεις που περιγράφουν το πρόβλημα είναι:

Για την προετοιμασία ενός δέματος i χρειαζόμαστε χρόνο:

$$t_i(q) = w_i \sum_k p_k\,, \qquad k: q(k) \le q(i)\ \ (\sigma\varepsilon\ \pi\lambda\acute{\eta}\theta o\varsigma\ \delta\varepsilon\mu\acute{\alpha}\tau\omega\nu)$$

Ο χρόνος που χρειάζεται ο υπάλληλος για να εξυπηρετήσει όλα τα δέματα:

$$Total(q) = \sum_{i=0}^{n} t_i(q)\,, \qquad k: q(k) \le q(i)\ \ (\sigma\varepsilon\ \pi\lambda\acute{\eta}\theta o\varsigma\ \delta\varepsilon\mu\acute{\alpha}\tau\omega\nu)$$

Παρατηρώντας το μοτίβο των αθροισμάτων του συνολικού χρόνου εξυπηρέτησης, συμπεραίνουμε εύκολα ότι όσο νωρίτερα δρομολογήσουμε ένα δέμα i, τόσες περισσότερες φορές ο χρόνος προετοιμασίας του, p_i, εμφανίζεται (σε σχέση με όλα τα υπόλοιπα p_i) στα αθροίσματα αυτά. Επιπλέον, τόσες λιγότερες φορές εμφανίζεται (σε σχέση με όλα τα υπόλοιπα w_i) το βάρος w_i στα αθροίσματα αυτά. Η αναλογία αυτή των τιμών p_i, w_i μας οδηγεί στο να σκεφτούμε τον εξής άπληστο αλγόριθμο:

Θα χρονοδρομολογούμε τα δέματα με βάση το μεγαλύτερο πηλίκο $\frac{w_i}{p_i}$.

Απόδειξη ορθότητας:

Θα χρησιμοποιήσουμε το επιχείρημα της ανταλλαγής.

Θέτουμε q^* τη βέλτιστη δρομολόγηση, q_g τη λύση του άπληστου και $Total(q^*)$, $Total(q_g)$ συνολικό χρόνο αντίστοιχα. Έστω ότι ο άπληστος αλγόριθμος δίνει διαφορετική λύση από την βέλτιστη. Έστω, ακόμα, ότι τα πρώτα $l-1$ δρομολογημένα δέματα και των δύο λύσεων είναι ίδια και το l-οστό δέμα της q_g (έστω d) δρομολογείται αργότερα στην q^*. Θα ισχύει για τους συνολικού χρόνους:

$$Total(q^*) = \cdots + w_{l-1} \sum_k p_k + \cdots + w_m \sum_k p_k + \ \ w_l \sum_k p_k + \cdots$$

$$Total(q_g) = \cdots + w_{l-1} \sum_k p_k + w_l \sum_k p_k + \cdots + w_m \sum_k p_k + \cdots$$

Αν ανταλλάσσαμε τις θέσεις των m και l δεμάτων στον βέλτιστο θα είχαμε:

$$Total(q^*{}_{new}) = \cdots + w_{l-1}\sum_k p_k + \cdots + w_l \sum_k p_k + w_m \sum_k p_k + \cdots$$

$$Total(q^*{}_{new}) - Total(q^*) = w_m p_l - w_l p_m$$

Στον άπληστο αλγόριθμο έχουμε ισχυριστεί ότι:

$$\frac{w_m}{p_m} \leq \frac{w_l}{p_l} \implies w_m p_l - w_l p_m \leq 0$$

Άρα όντως ο χρόνος μπορεί να βελτιωθεί με αυτήν την αλλαγή. Επομένως η άπληστη λύση είναι βέλτιστη.

Η χρονική πολυπλοκότητα θα είναι $O(n \log n)$ για την ταξινόμηση των στοιχείων $\frac{w_i}{p_i}$.

2.

Στην περίπτωση των δύο υπαλλήλων θα χρειαστούμε την βοήθεια του δυναμικού προγραμματισμού. Ακολουθώντας την λογική του προηγούμενου ερωτήματος, κάθε υπάλληλος θα πρέπει να χρονοδρομολογήσει την προετοιμασία των δεμάτων με βάση το μεγαλύτερο $\frac{w_i}{p_i}$. Επομένως ταξινομούμε τα δέματα με αυτόν τον τρόπο. Αν S τα δέματα που θα εξυπηρετήσει ο πρώτος υπάλληλος θα έχουμε την ακόλουθη αναδρομική σχέση για τα πρώτα i δέματα:

$$Total(i, S) = \min \begin{cases} Total(i-1, S \cup \{i\}) + w_i \left(\sum_k p_k + p_i \right), & k < i, k \in S \\ Total(i-1, S) + w_i \left(\sum_j p_j + p_i \right), & j < i, j \notin S \end{cases}$$

Όπου στην πρώτη περίπτωση επιλέγεται να εξυπηρετήσει το δέμα i ο πρώτος υπάλληλος ενώ στην δεύτερη ο δεύτερος. Με βάση την παραπάνω αναδρομή συμπληρώνουμε έναν πίνακα $n \times n$ από πάνω προς τα κάτω και στο στοιχείο με δείκτη $(n-1, n-1)$ θα βρίσκεται η λύση. Η χρονική πολυπλοκότητα είναι προφανώς $O(n^2)$. Για $x \geq 3$ υπαλλήλους, η πολυπλοκότητα αυξάνεται εκθετικά, συνεπώς στην αναδρομική σχέση θα έπρεπε να προσθέσουμε τις ποσότητες $S_2, S_3, \dots, S_{x-1}$ για τα δέματα που θα εξυπηρετήσουν οι $x - 1$ πρώτοι υπάλληλοι. Επομένως θα χρειαστούμε πίνακα x διαστάσεων. Η χρονική πολυπλοκότητα θα γινόταν τότε $O(n^x)$.

Άσκηση 3

(α)

Η βέλτιστη λύση δίνεται με τη βοήθεια δυναμικού προγραμματισμού. Η αναδρομική σχέση που περιγράφει τη βέλτιστη λύση είναι:

$$P(i) = min\left\{P(j-1) + \left(x_i - x_j\right)^2 + C\right\}, \qquad 0 \le j \le i$$

$P(i)$ είναι η ελάχιστη δυνατή τιμή του κόστους τοποθέτησης στεγάστρων στα σημεία $x_0, x_1, \dots, x_i$, με την παραδοχή ότι $P(-1) = 0$. Ξεκινώντας από το σημείο x_f, επιλύουμε αναδρομικά την παραπάνω σχέση, υπολογίζοντας για κάθε x_i το ελάχιστο κόστος $P(i)$ μεταξύ όλων των πιθανών συνδυασμών στεγών μέχρι και το σημείο αυτό. Η λύση μας θα είναι η $P\left(x_f\right)$.

Χρονική πολυπλοκότητα $O(n^2)$.

(β)

$$lines = \{(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)\}, \qquad 0 \ge a_1 \ge a_2 \ge \dots \ge a_n$$

$$points = \{x_1, x_2, \dots, x_k\}, \qquad x_1 \le x_2 \le \dots \le x_k$$

Αρκεί να βρούμε το κυρτό κάλυμμα των ευθειών. Εξετάζουμε τις ευθείες ξεκινώντας από την (a_n, b_n) (την προσθέτουμε στην λίστα $curve = [(a_n, b_n)]$ και ορίζουμε ως προηγούμενο σημείο τομής το $-\infty$). Προσθέτουμε και την αμέσως επόμενη ευθεία στη λίστα και αποθηκεύουμε το σημείο τομής των δυο αυτών ευθειών ($curve = [(a_n, b_n), (a_{n-1}, b_{n-1})]$). Στη συνέχεια για κάθε ευθεία i, για i από $n-2$ έως 1, αν το σημείο τομής της με την τελευταία ευθεία στη λίστα είναι μεγαλύτερο ή ίσο (ως προς τη συντεταγμένη y_i) από σημείο τομής των προηγούμενων δύο ευθειών της λίστας, τότε πρόσθεσε την ευθεία στην λίστα και συνέχισε. Αλλιώς αφαίρεση την τελευταία ευθεία της λίστας και επανέλαβε την διαδικασία. Τα σημεία τομής βρίσκονται σε σταθερό χρόνο, οπότε η χρονική πολυπλοκότητα για την συμπλήρωση της λίστας θα είναι $O(n)$.

Τώρα αρκεί να ελέγξουμε για κάθε x_j ποια ευθεία μας δίνει μικρότερη τιμή. Αυτό γίνεται εύκολα αν διατρέξουμε την λίστα $curve$ από το τέλος προς την αρχή ωσότου να ισχύει $a_{i+1}x_j + b_{i+1} > a_i x_j + b_i$.

Οπότε η ευθεία i θα δίνει ελάχιστο για το x_j και αφού $x_j \le x_{j+1}$, τότε αρκεί να ξεκινήσουμε την αναζήτηση για την ευθεία που δίνει ελάχιστο για το x_{j+1} από την ευθεία i ($O(n+k)$). Οπότε η συνολική χρονική πολυπλοκότητα θα είναι $\Theta(n+k)$.

Από το **(α)** έχουμε:

$$P(i) = min\left\{P(j-1) + \left(x_i - x_j\right)^2 + C\right\} \Longrightarrow$$

$$P(i) = min\{P(j-1) + {x_i}^2 - 2x_i x_j + {x_j}^2 + C\} \Longrightarrow$$

$$P(i) = min\{{x_i}^2 + \left(-2x_j\right)x_i + \left(P(j-1) + {x_j}^2\right) + C\} \Longrightarrow$$

$$P(i) = {x_i}^2 + C + min\left\{\underbrace{\left(-2x_j\right)}_{a_j} x_i + \underbrace{\left(P(j-1) + {x_j}^2\right)}_{b_j}\right\}$$

Μετατρέποντας με τον παραπάνω τρόπο την αναδρομική σχέση, συμπεραίνουμε πως μπορούμε να χρησιμοποιήσουμε την λύση του **(β)** για των υπολογισμό του $min\{a_j x_i + b_j\}$, όπου $a_j = \left(-2x_j\right), b_j = \left(P(j-1) + {x_j}^2\right)$. Άρα μπορούμε να λύσουμε το πρόβλημα του ερωτήματος **(α)** σε χρόνο $\Theta(n)$.

Άσκηση 4

Η λύση του προβλήματος επιτάσσει τη χρήση δυναμικού προγραμματισμού. Συγκεκριμένα θα ορίσουμε μία αναδρομική σχέση για το ελάχιστο συνολικό δείκτη ευαισθησίας $S(b,i)$, όπου βάζουμε i φοιτητές σε k λεωφορεία. Θα έχουμε, με αρχή το τελευταίο λεωφορείο:

$$S(b,i) = min\left\{S(b-1,j) + \sum_{x=j+1}^{i}\sum_{y=x+1}^{i} A_{xy}\right\}, \qquad j < i$$

Έστω ο πίνακας A $n \times n$ τέτοιος ώστε $A[x][y] = A_{xy}$.

Συνεπώς θα μπορούμε να υπολογίσουμε σε σταθερό χρόνο το παραπάνω διπλό άθροισμα ως εξής:

$$\sum_{x=j+1}^{i}\sum_{y=x+1}^{i} A_{xy} = \frac{1}{2}\left(\alpha(i,i) - \alpha(i,j) - \alpha(j,i) + \alpha(j,j)\right),$$

Όπου:

$$\alpha(e,m) = \alpha(e-1,m) + \alpha(e,m-1) - \alpha(e-1,m-1) + A[e][m]$$

Η χρονική πολυπλοκότητα θα είναι $O(kn^2)$.

Για να την επίλυση του δεύτερου σκέλους θα εφαρμόσουμε την Divide & Conquer Optimization μέθοδο. Ορίζουμε τα παρακάτω minimizers της αναδρομικής σχέσης προκειμένου να αποφύγουμε τον έλεγχο όλων των μαθητών $j < i$ στα ελάχιστα:

$$S_{min}(b,i) = argmin\left\{S(b-1,j) + \sum_{x=j+1}^{i}\sum_{y=x+1}^{i} A_{xy}\right\}, \qquad j < i$$

Συνεπώς $S_{min}(b,i) \in [1, i-1]$ είναι ο τελευταίος φοιτητής που δεν παίρνει το τελευταίο λεωφορείο της βέλτιστης λύσης για b λεωφορεία και i φοιτητές. Εύκολα μπορούμε να συμπεράνουμε ότι θα ισχύει $S_{min}(b,i) \le S_{min}(b,i+1)$, δηλαδή για περισσότερους φοιτητές και ίδιο αριθμό λεωφορείων με συμφέρει να αφήσω το τελευταίο σύνολο για αργότερα (ιδιότητα μονοτονίας). Επομένως θα ακολουθήσουμε τα εξής βήματα:

Αρχικές συνθήκες:

$S_{min}(1,i) = 0$

$S(1,i) = \frac{1}{2}a(1,i)$, για $i = 1, \dots, n$

Θα υπολογίσουμε κάθε γραμμή ℓ των πινάκων S_{min}, S, εκμετεαλλευόμενοι την μονοτονία των $S_{min}(b,i)$. Δεδομένης της $\ell - 1$ θα έχουμε για την ℓ:

$$S_{min}\left(b,\frac{n}{2}\right) = argmin\left\{S(b-1,j) + \sum_{x=j+1}^{i}\sum_{y=x+1}^{\frac{n}{2}} A_{xy}\right\} = C, \qquad j < \frac{n}{2}$$

$$S_{min}\left(b,\frac{n}{4}\right) = argmin\left\{S(b-1,j) + \sum_{x=j+1}^{i}\sum_{y=x+1}^{\frac{n}{4}} A_{xy}\right\}, \qquad j < min\left(\frac{n}{4}, C\right)$$

$$S_{min}\left(b,\frac{3n}{4}\right) = argmin\left\{S(b-1,j) + \sum_{x=j+1}^{i}\sum_{y=x+1}^{\frac{3n}{4}} A_{xy}\right\}, \qquad C \le j < min\left(\frac{3n}{4}, C\right)$$

...

Υπολογίζουμε έτσι κάθε γραμμή σε $O(n \log n)$. Για k γραμμές θα έχουμε $O(kn \log n)$. Επιπλέον θέλουμε χρόνο $O(n^2)$ για τον υπολογισμό των τελικών αθροισμάτων.

Συνεπώς συνολικά θα έχουμε χρονική πολυπλοκότητα $O(n^2 + kn \log n)$.

Περαιτέρω βελτίωση μπορεί να επιτευχθεί με χρήση του Knuth Optimization, λαμβάνοντας υπόψιν την πιο αυστηρή ιδιότητα της μονοτονίας:

$$S_{min}(b-1,i) \leq S_{min}(b,i) \leq S_{min}(b,i+1)$$

Με αυτόν τον τρόπο υπολογίζουμε όλους τους minimizers με σταθερή διαφορά $b-i$ σε συνολικό χρόνο $O(i)$.

Συνολική πολυπλοκότητα: $O(n^2)$

Άσκηση 5

(α)

Θεωρούμε ένα συνεκτικό μη κατευθυνόμενο γράφημα $G(V,E)$.

Έχουμε:

T_1, T_2 δύο διαφορετικά δέντρα με τουλάχιστον μία κοινή ακμή και $e' \in T_2, \backslash T_1$.

Εύκολα συμπεραίνουμε πως το δέντρο $T_1 \cup \{e'\}$ περιέχει κύκλο, ενώ ακόμα θα υπάρχει κάποια ακμή e που ανήκει στον κύκλο, τέτοια ώστε $e \in T_1, \backslash T_2$, αφού αν $e \in T_2$ τότε το T_2 θα είχε κύκλο. Συνεπώς το δέντρο $(T_1 \backslash \{e\}) \cup \{e'\}$ δεν θα περιέχει κύκλο και θα έχει πλήθος ακμών $n-1$. Άρα θα αποτελεί συνδετικό δέντρο.

Αλγόριθμος εύρεσης e':

$T_1 \backslash \{e\} \longrightarrow T_{1a}, T_{1b}$, δύο συνεκτικά δέντρα που δημιουργούνται με την αφαίρεση της ακμής e.

Δημιουργούμε τα set (hash table υλοποίηση) εκτελώντας αναζήτηση κατά βάθος στα T_2, T_{1a} αντίστοιχα (μπορούσαμε και στο T_{1b} αντί του T_{1a}):

$$E_{T_2} = \{\alpha\kappa\mu\acute{\epsilon}\varsigma\ \tau o\upsilon\ T_2\}, \qquad V_{T_{1a}} = \{\kappa o\rho\upsilon\varphi\acute{\epsilon}\varsigma\ \tau o\upsilon\ T_{1a}\}$$

Εκτελούμε αναζήτηση κατά βάθος $\big(O(|V|+|E|)\big)$ στον Γράφο και βρίσκουμε όλες τις ακμές (u,v), τέτοιες ώστε $u \in V_{T_{1a}}$ και $v \notin V_{T_{1a}}$ (έλεγχος σε $O(1)$). Αν $(u,v) \in E_{T_2}$ (έλεγχος σε $O(1)$) τότε η λύση μας είναι $e' = (u,v)$. Αλλιώς συνεχίζουμε την επανάληψη.

Χρονική Πολυπλοκότητα: $O(|V|+|E|)$.

(β)

Έστω το γράφημα H και $T_1, T_2 \in H$ συνεκτικά δέντρα του $G(V, E)$.

Κάθε δέντρο $T_i \in H$ θα έχει V κορυφές και $V-1$ ακμές και θα είναι μοναδικό.

Θα δείξουμε ότι το H είναι συνεκτικό και επιπλέον η ελάχιστη απόσταση $d_{1,2}$ των T_1, T_2 (στο Η) είναι $d_{1,2} = |T_1 \backslash T_2|$.

Θα δουλέψουμε επαγωγικά:

Υπόθεση: $|T_1 \backslash T_2| = m$ αν και μόνο αν $d_{1,2} = m$

Γνωρίζουμε ότι για τα T_1, T_2 συνδέονται με ακμή **μόνο** αν $|T_1 \backslash T_2| = 1$. Άρα $d_{1,2} = 1 = |T_1 \backslash T_2|$.

Αν τώρα έχουμε $e \in T_1 \backslash T_2$, τότε από **(α)** θα ισχύει $T_3 = (T_1 \backslash \{e\}) \cup \{e'\} \in H$

$(e' \in T_2, \backslash T_1)$.

Έστω ότι $|T_3 \backslash T_2| = m$. Τα T_3, T_2 θα έχουν $V-1-m$ κοινές ακμές, ενώ τα T_3, T_1 θα έχουν $V-1-1$ κοινές ακμές. Άρα $|T_1 \backslash T_2| = m+1$.

Εφόσον $d_{1,2} = m+1$, $\exists T_3 \in H$ τέτοιο ώστε $d_{1,3} = 1, d_{2,3} = m$. Από υπόθεση $|T_3 \backslash T_2| = m$, οπότε $|T_1 \backslash T_2| = m-1$ ή $|T_1 \backslash T_2| = m+1$. Αν $|T_1 \backslash T_2| = m-1$ τότε $d_{1,2} = m-1$, άτοπο.

Ο αλγόριθμος για το συντομότερο μονοπάτι μεταξύ δύο δέντρων $T_1, T_2 \in H$ (στο H) θα είναι ο εξής:

Ισχύει: $|T_1 \backslash T_2| = |T_2 \backslash T_1|$.

Υπολογίζουμε αρχικά όλες τις ακμές που ανήκουν στο T_2 αλλά όχι στο T_1 $(T_2 \backslash T_1)$ και το αντίστροφο $(T_1 \backslash T_2)$. Στην συνέχεια για κάθε ακμή $e' \in T_2 \backslash T_1$ επιλέγουμε (κάθε φορά ξεχωριστή) μία ακμή $e \in T_1 \backslash T_2$, και θέτουμε: $T_1 = (T_1 \backslash \{e\}) \cup \{e'\}$.

Αν $T_2 \backslash T_1 = m$ τότε ο αλγόριθμος θα εκτελέσει m βήματα προκειμένου να μετατρέψει το T_1 σε T_2. Κάθε ενδιάμεση μετατροπή του T_1 αποτελεί κομμάτι του βέλτιστου μονοπατιού.

Η χρονική πολυπλοκότητα θα είναι :

Βρίσκουμε σε $O(2|V|)$ τα παραπάνω σύνολα ακμών (έστω ο έλεγχος $e \in T_1, e' \in T_2$ γίνεται σε $O(1)$). Τέλος, Οι ανανεώσεις των ακμών θα κοστίζουν χρόνο $O(|V|)$ για m επαναλήψεις. Άρα συνολικά $O(m \cdot |V|)$.

(γ)

Για την επίλυση του προβλήματος θα ακολουθήσουμε τα εξής βήματα:

Βρίσκουμε αρχικά το ελάχιστο συνεκτικό δέντρο T με χρήση του αλγορίθμου του Kruskal ($O(m \cdot \log m)$), καθώς και το συνολικό βάρος του $w(T)$. Για κάθε ακμή $e = (x, y)$ που ελέγχουμε η οποία δεν ανήκει στο T, την προσθέτουμε στο δέντρο και από τον κύκλο που δημιουργείται αφαιρούμε την ακμή με το μεγαλύτερο βάρος (διαφορετική φυσικά από αυτήν που προσθέσαμε). Έτσι, λαμβάνουμε ένα νέο ελάχιστο συνεκτικό δέντρο T' που περιέχει την ελεγχόμενη ακμή. Η ορθότητα του επιχειρήματος αυτού είναι εύκολο να την αντιληφθούμε διαισθητικά αν λάβουμε υπόψιν την λογική του Kruskal.

Αντί να εντοπίζουμε κάθε φορά τον κύκλο, αρκεί να βρούμε το μονοπάτι που ενώνει τις κορυφές x, y στο T για την αντίστοιχη ακμή $e = (x, y)$ που προσθέτουμε, μιας και η ακμή που θα αφαιρέσουμε θα ανήκει σε αυτό το μονοπάτι. Με αναζήτηση κατά βάθος, υπολογίζουμε και αποθηκεύουμε, εκ των προτέρων, για κάθε ζεύγος κορυφών x, y το βάρος

$w_{x,y}(T)$ της βαρύτερης ακμής στο μονοπάτι του T που τις ενώνει. Έτσι το τελικό ζητούμενο βάρος για κάθε ακμή θα δίνεται από τη σχέση: $w(T_e) = w(T) - w_{x,y}(T) + w(e)$